化工应用数学 第三章 数据处理 讲义

（01.插值算法）

插值：

插值是离散函数逼近的重要方法，利用它可通过函数在有限点的取值状况，估算出函数在其他点的近似值。

表 2 硫酸锰溶解度（来自《化学化工物性手册》P423）

温度	0	10	20	30	40	50	60	70	80	90	100
溶解度	52.9	—	62.9	62.9	60	—	53.6	—	45.6	36.4	28.8
温度	120	130	140	150	160	170	180				
溶解度	21.0	13.5	8.93	5.6	3.4	2.09	1.26				

每 100g 水中能溶解的硫酸锰的质量。

我们在化工手册上查看的各种物质物性数据以及在数学手册上查到的各种函数取值表等，都是类似于上图中的形式。它们一般都是给出了某些参数下的数值，如上表中的温度，当我们需要的数据没有在表上如 56°，这时就需要使用到插值算法。

拉格朗日插值：

若已知函数 y=f(x)在[a, b]上有定义，且有 n+1 个互异的值 $x_0,x_1,\cdots,x_n$ 处的函数值为 $y_0,y_1,\cdots,y_n$,则可以构造一个过这 n+1 个点的、次数不超过 n 的多项式 $y=P_n(x)$,使其满足：

$$P_n(x_k) = y_k, k \in [0, n]$$

函数 $y=P_n(x)$称为 f(x)的插值函数，我们可以使用 $P_n(x)$去求得不同位置处函数 f(x)的值，一般的 $P_n(x)$可以写作如下拉格朗日多项式的形式：

$$P_n(x) = \sum_{j=0}^{n} y_j P_j(x)$$

下面我们以三个点为例来讨论这个问题，即已知量为三个点(x_1,y_1)、(x_2,y_2)和(x_3,y_3)，这时我们可以合理的假设插值函数/拉格朗日多项式具有二次多项式的形式：

$$y = a_0 + a_1 x + a_2 x^2$$

一般的，我们很容易想到可以把已知点的坐标带入上述方程去求出相应的参数，即：

$$\begin{cases} y_1 = a_0 + a_1 x_1 + a_2 x_1^2 \\ y_2 = a_0 + a_1 x_2 + a_2 x_2^2 \\ y_3 = a_0 + a_1 x_3 + a_2 x_3^2 \end{cases}$$

但要注意，当我们已知的点更多的时候，上述方程组的复杂度也会相应增多，不便于我们去计算，下面我们来看拉格朗日是怎么处理这个问题的。

拉格朗日的思路：

1.这肯定是二次曲线

2.这条二次曲线可以根据三条二次曲线相加得到，它们满足：

第一根曲线 $f_1(x)$，在 x_1 点处，取值为 1，其余两点取值为 0

第二根曲线 $f_2(x)$，在 x_2 点处，取值为 1，其余两点取值为 0

第三根曲线 $f_3(x)$，在 x_3 点处，取值为 1，其余两点取值为 0

此时：

$$f(x) = y_1 f_1(x) + y_2 f_2(x) + y_3 f_3(x)$$

可以很容易的证明：

$y_1 f_1(x)$可以保证，在 x_1 点处，取值为 y_1，其余两点取值为 0

$y_2 f_2(x)$可以保证，在 x_2 点处，取值为 y_2，其余两点取值为 0

$y_3 f_3(x)$可以保证，在 x_3 点处，取值为 y_3，其余两点取值为 0

从而保证 $f(x)$通过三个点(x_1,y_1)、(x_2,y_2)和(x_3,y_3)。

我们可以容易的发现，下面的 $f_1(x)$是满足的：

$$f_1(x) = \frac{(x - x_2)(x - x_3)}{(x_1 - x_2)(x_1 - x_3)}$$

同样，对于 $f_1(x)$、$f_1(x)$我们也可以同样的构造出来，这三个函数可以统一为：

$$f_i(x) = \prod_{j \neq i}^{1 \le j \le 3} \frac{(x - x_j)}{(x_i - x_j)}$$

更一般的，对于任意 n 值来说，我们可以推导得到：

$$L_n(x) = \sum_{j=0}^{n} y_j P_j(x)$$

$$P_k(x)=\prod_{i\in B_k}\frac{x-x_i}{x_k-x_i}$$

其中 $B_k=\{i|i\neq k,i\in D_n\}$，$D_n=\{0,1,\cdots,n\}$

拉格朗日插值的程序实现：

使用 scipy.interpolate 模块下的 lagrange 函数：

```
from scipy.interpolate import lagrange
x=[1,2,3,4]
y=[1,16,26,36]
res=lagrange(x,y)
print(res)
```

显示结果为：

```
          3         2
0.8333 x - 7.5 x + 31.67 x - 24
```

线性插值：

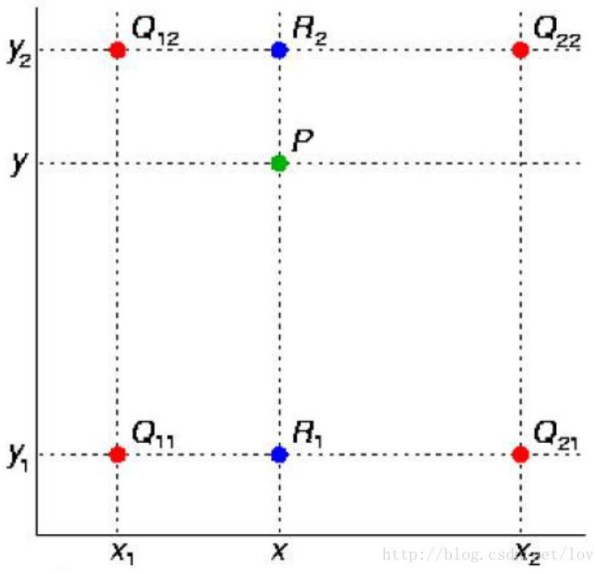
已知数据 (x0, y0) 与 (x1, y1)，要计算 [x0, x1] 区间内某一位置 x 在直线上的 y 值：

$$\frac{y-y_0}{x-x_0}=\frac{y_1-y_0}{x_1-x_0}$$

$$y=\frac{x_1-x}{x_1-x_0}y_0+\frac{x-x_0}{x_1-x_0}y_1$$

双线性插值：

双线性插值是有两个变量的插值函数的线性插值扩展，其核心思想是在两个方向分别进行一次线性插值，比如下图：



首先得到点 R1、R2 的值：

$$f(R_1) \approx \frac{x_2 - x}{x_2 - x_1} f(Q_{11}) + \frac{x - x_1}{x_2 - x_1} f(Q_{21}) \quad \text{where} \quad R_1 = (x, y_1)$$

$$f(R_2) \approx \frac{x_2 - x}{x_2 - x_1} f(Q_{12}) + \frac{x - x_1}{x_2 - x_1} f(Q_{22}) \quad \text{where} \quad R_2 = (x, y_2)$$

进一步的可以得到中间点 P 的值：

$$f(P) \approx \frac{y_2 - y}{y_2 - y_1} f(R_1) + \frac{y - y_1}{y_2 - y_1} f(R_2).$$

$$f(x,y) \approx \frac{f(Q_{11})}{(x_2 - x_1)(y_2 - y_1)}(x_2 - x)(y_2 - y) + \frac{f(Q_{21})}{(x_2 - x_1)(y_2 - y_1)}(x - x_1)(y_2 - y)$$

$$+ \frac{f(Q_{12})}{(x_2 - x_1)(y_2 - y_1)}(x_2 - x)(y - y_1) + \frac{f(Q_{22})}{(x_2 - x_1)(y_2 - y_1)}(x - x_1)(y - y_1).$$

本质上，我们可以将上式看作是一种面积平均的概念。

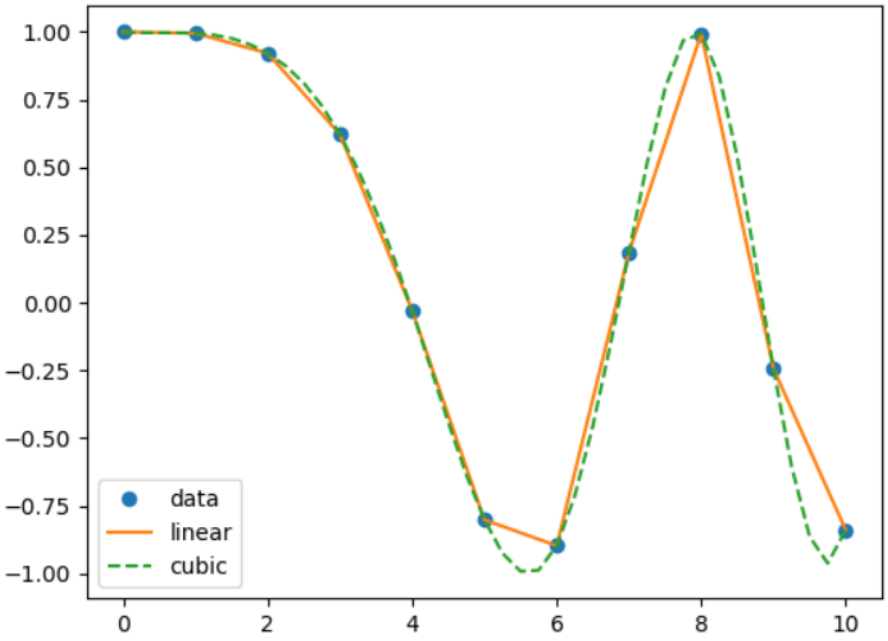
Scipy 一维插值：

interp1d：scipy.interpolate 包里有很多的模块可以实现对一些已知的点进行插值，即找到一个合适的函数，例如模块 interp1d。当得到插值函数后便可用这个插值函数计算其他 xj 对应的的 yj 值了,这也就是插值的意义所在。

```
import numpy as np
from scipy.interpolate import interp1d
x = np.linspace(0, 10, num=11, endpoint=True)
y = np.cos(-x**2/10.0)
f = interp1d(x, y)
f2 = interp1d(x, y, kind='cubic')
xnew = np.linspace(0, 10, num=41, endpoint=True)
import matplotlib.pyplot as plt
plt.plot(x, y, 'o', xnew, f(xnew), '-', xnew, f2(xnew), '--')
plt.legend(['data', 'linear', 'cubic'], loc='best')
plt.show()
```

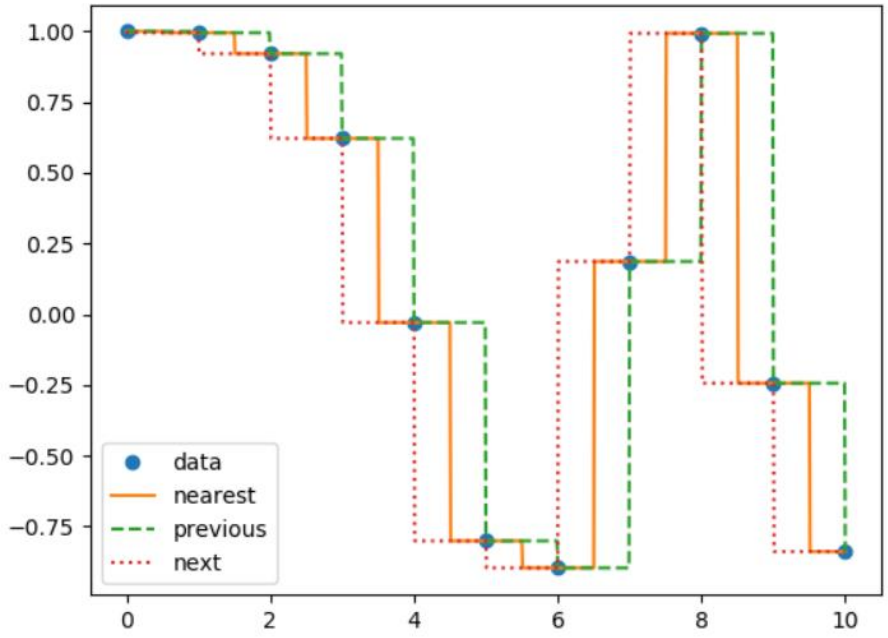
注意这里 interp1d 的返回值为一个函数，在使用 interp1d 之后我们能够直接将需要得到的参数参入 interp1d 的返回值，从而计算我们说需要的插值结果。

从这里可以对比不同的插值方式在 interp1d 表现出来的区别：



同时，这个函数还有很多不同的参数，下面给出一些常见的参数，如下图：

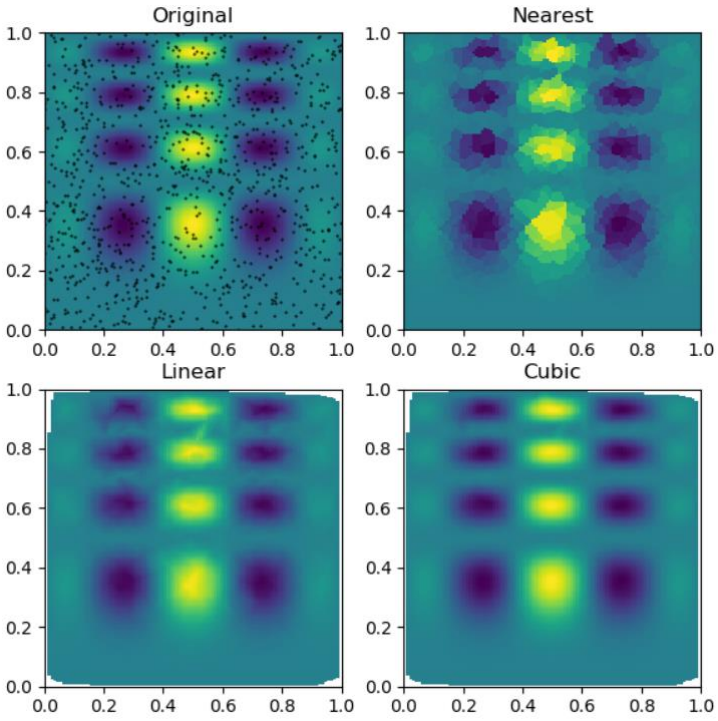
```
import numpy as np
from scipy.interpolate import interp1d
x = np.linspace(0, 10, num=11, endpoint=True)
y = np.cos(-x**2/10.0)
f1 = interp1d(x, y, kind='nearest')
f2 = interp1d(x, y, kind='previous')
f3 = interp1d(x, y, kind='next')
xnew = np.linspace(0, 10, num=1001, endpoint=True)
import matplotlib.pyplot as plt
plt.plot(x, y, 'o')
plt.plot(xnew, f1(xnew), '-', xnew, f2(xnew), '--', xnew, f3(xnew), ':')
plt.legend(['data', 'nearest', 'previous', 'next'], loc='best')
plt.show()
```



Scipy 二维插值：

Scipy 提供了 griddata 函数来进行二维问题的插值计算

```
import numpy as np
def func(x, y):
    return x*(1-x)*np.cos(4*np.pi*x) * np.sin(4*np.pi*y**2)**2
grid_x, grid_y = np.mgrid[0:1:100j, 0:1:200j]
points = np.random.rand(1000, 2)
values = func(points[:,0], points[:,1])
from scipy.interpolate import griddata
grid_z0 = griddata(points, values, (grid_x, grid_y), method='nearest')
grid_z1 = griddata(points, values, (grid_x, grid_y), method='linear')
grid_z2 = griddata(points, values, (grid_x, grid_y), method='cubic')
import matplotlib.pyplot as plt
import numpy as np
from scipy.interpolate import interp1d
plt.subplot(221)
plt.imshow(func(grid_x, grid_y).T, extent=(0,1,0,1), origin='lower')
plt.plot(points[:,0], points[:,1], 'k.', ms=1)
plt.title('Original')
plt.subplot(222)
plt.imshow(grid_z0.T, extent=(0,1,0,1), origin='lower')
plt.title('Nearest')
plt.subplot(223)
plt.imshow(grid_z1.T, extent=(0,1,0,1), origin='lower')
plt.title('Linear')
plt.subplot(224)
plt.imshow(grid_z2.T, extent=(0,1,0,1), origin='lower')
plt.title('Cubic')
plt.gcf().set_size_inches(6, 6)
plt.show()
```



上面给出了相应的程序代码及运算的结果，可以对比发现，与 interp1d 函数类似，采用不同的方法会对结果有很大的影响。

Scipy 模块：

官网：https://www.scipy.org/

SciPy 是一款方便、易于使用、专为科学和工程设计的 Python 工具包。它包括统计、优化、整合、线性代数模块、傅里叶变换、信号和图像处理、常微分方程求解器等等。

Scipy 库构建于 NumPy 之上，提供了一个用于在 Python 中进行科学计算的工具集，如数值计算的算法和一些功能函数，可以方便的处理数据。主要包含以下内容：

- 特殊函数 (scipy.special)
- 积分 (scipy.integrate)
- 最优化 (scipy.optimize)
- 插值 (scipy.interpolate)
- 傅立叶变换 (scipy.fftpack)
- 信号处理 (scipy.signal)
- 线性代数 (scipy.linalg)
- 稀疏特征值 (scipy.sparse)
- 统计 (scipy.stats)
- 多维图像处理 (scipy.ndimage)
- 文件 IO (scipy.io)